Project 2: Analysis of Anonymous Algorithms

Project Overview

This project involves analysis of several anonymous algorithms written in pseudo-code. You are to:

- Analyze the run time (best case and worst case) of these algorithms using the techniques discussed in class.
- Make predictions of the run time for various sized problems, for both best and worst case. Use these estimates to design your test cases.
- Implement the algorithms in your chosen programming language (C++, Java, Python).
- Test the implementation. Measure the running time for enough problem sizes that you can get reliable results.
- Compare the measured run time to your predicted run time and explain any discrepancies.
- Finally, determine what each algorithm *does* - what does it compute, what process does it implement, why would you use this algorithm.

Algorithms Definitions and Descriptions

In each of these algorithms, there are several common variables, especially in the function parameters.

- Variables named A, B, C, D, E are arrays of integers. All arrays are indexed beginning at 1.
- Variables named i, j, k, l, m, n, p, q, r are integers.
- The variable N is the size of the initial array passed to the algorithm.
- When creating your initial array A of size N, you may assume that all elements are integers in the range $1 \ldots N$. You may have repeats, but no values outside this range.

Project 2: Analysis of Anonymous Algorithms

Algorithm 1

For Algorithm 1, make use of the general description above

```
1   Algorithm1(A)
2       B[1..N] ← 0
3       Compute(A,B,1)
4       return
5
6   Compute(A,B,i)
7       if i > N
8           Dump(B)
9           return
10      B[i] ← 0
11      Compute(A,B,i+1)
12      B[i] ← A[i]
13      Compute(A,B,i+1)
14      B[i] ← 0
15      return
16
17  Dump(B)
18      for j ← 1 to N
19          if B[j] ≠ 0
20              Print B[i]
21      return
```

Algorithm 2

In this algorithm, the variable Z is an array of linked lists. These lists can be traversed in the usual way. Division is assumed to be integer division (i.e. it drops the fractional part). Also the % sign is a modulus operator - it takes the remainder.

```
1   Algorithm2(A)
2       Compute(A)
3       Dump(A)
4       return
5
6   Compute(A)
7       size ← ⌊√N⌋ + 1
8       Z[1..size] ← EmptyList
9       for i ← 1 to N
10          append A[i] to end of Z[A[i] mod size]
11      counter ← 1
12      for i ← 1 to size
13          while Z[i] is not empty
14              A[counter] ← FirstElement(Z[i])
15              counter++
16              DeleteFirstElement(Z[i])
17      for i ← 1 to N
18          append A[i] to end of Z[A[i]/size]
19      counter ← 1
20      for i ← 1 to size
21          while Z[i] is not empty
22              A[counter] ← FirstElement(Z[i])
23              counter++
24              DeleteFirstElement(Z[i])
25      return
26
27  Dump(B)
28      for i ← 1 to N
29          print B[i]
30      return
```

Project 2: Analysis of Anonymous Algorithms

Algorithm 3

```
1  Algorithm3(A)
2      B[1..N] ← 1
3      C[1..N] ← −1
4      Compute(A,B,C)
5      m ← 1
6      n ← 0
7      for i ← 1 to N
8          if m < B[i]
9              m ← B[i]
10             n ← i
11     Dump(A,C,n)
12     return
13
14 Compute(A,B,C)
15     for i ← 2 to N
16         m ← 0
17         n ← −1
18         for j ← i−1 to 1
19             if A[j] < A[i]
20                 if B[j] > m
21                     m ← B[j]
22                     n ← j
23         B[i] ← m + 1
24         C[i] ← n
25     return
26
27 Dump(A,C,i)
28     if i < 1
29         return
30     Dump(A,C,C[i])
31     Print A[i]
32     return
```

Algorithm 4

The function *random_between(p,r)* will return a random integer between p and r, inclusive (i.e. both p and r are possible values). This function executes in constant (i.e. $\Theta(1)$) time.

```
1  Algorithm4(A)
2      i ← (N+1)/2
3      val ← Compute(A,1,N,i)
4      Dump(val)
5
6  Compute(A,p,r,i)
7      if p = r
8          return A[p]
9      n ← random_between(p,r)
10     swap(A[n],A[r])
11     n ← A[r]
12     q ← p−1
13     for j ← p to r−1
14         if A[j] ≤ n
15             q ← q+1
16             swap(A[q],A[j])
17     q ← q+1
18     swap(A[q],A[r])
19     k ← q−p+1
20     if i = k
21         return A[q]
22     else if i < k
23         return Compute(A,p,q−1,i)
24     else
25         return Compute(A,q+1,r,i−k)
26
27 Dump(val)
28     Print val
```

Project 2: Analysis of Anonymous Algorithms

Algorithm 5

In this algorithm, A is an array of integers and N is the size of the array. Indexing begins at 1.

```
1  Algorithm5(A,N)
2     B ← Array[N]
3     j ← 1
4     i ← 1
5     while j ≤ N−1
6         i ← 1
7         while i ≤ N−j
8             p ← i
9             m ← i+j−1
10            r ← min(i−1+2*j, N)
11            for u ← m+1 to p by −1
12                B[u−1] ← A[u−1]
13            for v ← m to r−1
14                B[r+m−v] ← A[v+1]
15            for w ← p to r
16                if B[v] < B[u]
17                   A[w] ← B[v−−]
18                else
19                   A[w] ← B[u++]
20            i ← i + 2*j
21        j ← 2*j
22    return
```

Algorithm 6

In the following algorithm, A is an array, i is an index and N is the size of the array. The main program should call to ``Algorithm6(A,1,N)" where N is the size of the array.

```
1  Algorithm6(A, i, N)
2      if i = N
3          for j ← 1 to N−1 by 1
4              if A[j] > A[j+1]
5                  return false
6          return true
7      if (Algorithm6(A,i+1,N))
8          return true
9      for j ← i+1 to N by 1
10         swap(A[i],A[j])
11         if (Algorithm6(A,i+1,N))
12            return true
13         swap(A[i],A[j])
14     return false
```

Algorithm 7

In the following, A is the array and N is the size of the array.

```
1  Algorithm7(A,N)
2      B ← Array[N]
3      B[1..N] ← 0
4      for i ← 1 to N
5         B[A[i]]++
6      i ← 1
7      for j ← 1 to N
8         for k ← 0 to B[j]
9             A[i] ← j
10            i++
11     return
```

Algorithm 8

```
1  Algorithm8(A,N)
2     i ← 1
3     j ← N
4     while (i < j)
5         for k ← i to j−1
6             CompSwap(A[k],A[k+1])
7         for k ← j−1 to i+1 by −1
8             CompSwap(A[k−1],A[k])
9         i++
10        j−−
11    return
12
13 ▷ CompSwap passes by reference.
14 CompSwap(x,y)
15     if x > y
16        swap(x,y)
```

Project 2: Analysis of Anonymous Algorithms

Project Documentation Submission

Your written assessment for this project is a formal report describing your analysis of the above algorithms, your implementation, and a table and graphs presenting comparisons between your theoretically predicted run-time and your measured run-time. It is expected that your papers will contain the following information:

- A cover page with Assignment Title, Name, Date, Course, Instructor.
- A general description of what algorithms is and why it is important to measure algorithms using algorithmic analysis techniques.
- An introduction explaining the general approach you took toward analyzing, implementing and determining the function of each algorithm. The introduction should not discuss any of the individual algorithms in detail, but should cover your overall approach to the problem - e.g., what general procedures did you follow when trying to figure out what an algorithm does?
- For each algorithm, a section which contains your theoretical analysis. For example, this section might contain a simplification of the algorithm for analysis purposes, where you eliminate all irrelevant lines of code to show the execution structure of the algorithm. For example, if we were examining binary search (Hint: It is not one of the algorithms), you might simplify the algorithm to:

  binary_search(A,i,j)
  Base Case - T(1) = 1
  O(1) // determine midpoint
  binary_search(A,i,midpoint)

  where you have eliminated all of the exact computations and instead left the structure of the algorithm - binary search computes the midpoint and then determines which half of the array to search farther - the algorithm itself contains two recursive calls, but only one executes, so for analysis purposes, it can be reduced to the above simplified version. This section should contain your derivation of any recurrences involved as well as the solution.
- For each algorithm, you should predict the run times for several (at least 10) values of N. You will have to do this several times - your first predictions after deriving the run time should be aimed at determining what range of N to test. For example, if your algorithm turns out to be $\Theta(N^4)$(Hint: none of them are), you might initially compute for N = 100, 1000, 100000 and see which gives you a reasonable expected running time. As a initial approximation, you can assume that most computers can execute very roughly 1,000,000 to 10,000,000 complex operations per second. So, our first set of predictions are that N=100 would take about 1 second. This is right in the range for reasonable running times - running times of about 5-10 seconds will typically (but not always!) get you into the asymptotic region. This means that you could design your tests so that you are testing values of N in the vicinity of 50 to 500, rather than 10,000. A sample selection of target 100, 125, 150, 175, 200, 225, 250, 275, 300, 325 which would have a final expected running time of about 110 seconds.

Project 2: Analysis of Anonymous Algorithms

You will probably have to adjust these initial predictions once you have a working implementation. Run your program for one of your values of N and see how long the program actually takes. Adjust your final set of test values accordingly. For example, suppose that you implement an N^4 algorithm and run it with N=200 and find that it takes 25 seconds. Then N=325 would take about 175 seconds rather than our first estimate of 110. This is still reasonable, but 3 minutes is a long time to wait for a result if you have to run it several times so, while you probably would not change the test sequence, you might reduce it a little. However, a run with N=100, for example, might take 15 seconds instead of about 1, then you could be inclined to use smaller values of N for testing (since, in this case, N=325 would be expected to take 1673 seconds (about 1/2 hour).

- **(20 points)** For each algorithm you should have a brief discussion of any issues you came across while implementing the algorithm (for example, how do you deal with the random number generation in algorithm 4?). You should include a discussion of how you tested the algorithm, what kinds of inputs you gave it (sorted arrays, random arrays, arrays with all the same number, etc.) to see how (a) the algorithm output and (b) the run time depend on the input.
- **(20 points)** For each algorithm you should discuss what the algorithm does and how you came to that conclusion. Give supporting evidence, describe the tests you did which led you to that conclusion, describe any external sources which helped you and if you come across the algorithm in some external source, describe where you found.
- **(20 points)** For each algorithm you should present your predicted run times as a function of N, your measured run times and a comparison between them. The easiest way to get a meaningful comparison is to compute your predictions, measure the actual times, then plot the measured times vs the predicted times. If you have a correct analysis, this should produce a straight line. (Hint: if you have a very fast algorithm, you might have to use a log plot instead of a linear one.)

  There are two alternate methods - first, divide your measured quantities by the predicted and plot these vs ***N*** - the result should be approximately constant. Finally, divide your measured values by your predicted and average the results to get a ``scale factor''. Multiply all your measured values by the scale factor and plot both the scaled measured values and the predicted values as a function of N on the same graph. The points should follow the same curve and have approximately the same values.
- **(10 points)** For each algorithm, discuss your analysis and explain any significant discrepancies between your predicted run times and your actual run times.
- **(10 points)** Finally, you should have a bibliography. You will be able to find a lot of information on algorithms on the internet, but you should also check this information against more reliable sources. These sources should be cited in your text and listed in the bibliography.
- **(20 points) You must upload the source code for each algorithm.**